# Design and Implementation of a Genetic-Based Algorithm for Data Mining

Sunil Choenni

National Aerospace Laboratory NLR, P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
and
University of Twente, Dept. of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands
email: choenni@nlr.nl and choenni@cs.utwente.nl

## Abstract

Many data mining problems can be considered as search problems. A database is regarded as a search space and a mining algorithm as a search strategy. The search spaces that rise from data mining problems are very large, making an exhaustive search infeasible. Therefore, heuristic search strategies are of vital importance. In this paper, we discuss the design and implementation of a (prototype) data mining tool that is equipped with a genetic algorithm. We have mined two real-life aircraft incident databases with this tool. We report on the obtained mining results as well.

## 1 Introduction

Research and development in data mining evolves in several directions which are not necessarily divergent. One of these directions is the induction of classification rules from databases [1, 2, 13, 14, 17]. Many data mining problems in this direction can be regarded as search problems. A search problem is characterised by a search space and a search strategy [12]. A search strategy is used to identify specific elements in the search space by walking efficiently through this space.

In the context of data mining, a database is regarded as a set of tuples, and each (projected) subset of tuples is considered as an element in the search space. The problem is to select interesting subsets without inspecting the whole search space, which is the task of a search strategy. For example, in a car insurance environment the identification of profiles of risky drivers, i.e., drivers with (more than average) chances of causing an accident, can be modelled as a search problem. Consider an artificial relation *Driver(gender, age, town, category, price, damage)*, in which the attributes *gender*, *age*, and *town* refer to the driver and the other attributes refer to the car. Attribute *category* records, whether a car is leased or not, and *damage* records, whether a car has been involved in an accident or not. The challenge is to select a conjunction of predicates that represents the group of risky drivers. Assume that young males in leased cars form the group of risky drivers. Then, an expression like: *gender* **is** ('male') $\wedge$ *age* **in** [19,24] $\wedge$ *category* **is** ('leased') should be searched for.

In general, the search spaces that should be inspected in order to answer mining questions are very large, making an exhaustive search infeasible. Therefore, heuristic search strategies are of vital importance to data mining. The success of a search strategy is often dependent on the structure of the search space. For example, a hill climber will generally perform better on a search that consists of a few optima, while a genetic algorithm will perform better if the search space consists of many optima. The reason is that a hill climber terminates if it reaches an optimum, while a genetic algorithm does not. In a search space that contains many optima, it will generally be worthwhile to continue searching the space after having found the first optimum. Unfortunately, the search spaces that stem from data mining problems neither have a specific structure nor are the structures known in advance. On the basis of evidence, one should choose for a search algorithm. Therefore, a data mining tool should be equipped with several search algorithms.

This paper is devoted to the design and implementation of a (prototype) data mining tool, called SHARVIND, which is equipped with a genetic algorithm. SHARVIND is primarily developed for data

**Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.**

mining problems that give rise to search spaces that consist of expressions. An expression is a conjunction of predicates and each predicate is defined on a database attribute. A genetic algorithm is brought into action to efficiently search for interesting expressions.

In general, a genetic algorithm is characterised by the representation of individuals, a fitness function that evaluates an individual, and the manipulation operators cross-over and mutation [15]. We represent an individual as an expression. This representation fits seamlessly in the field of databases. The fitness function takes care that extracted knowledge from the database is supported by a significant part of the database and that trivial knowledge is discarded before-hand. Furthermore, our fitness function gives rise to the reduction of the number of disk accesses during the mining process. We have implemented the mutation operator such that an expression undergoes a minor modification. The cross-over operator takes two expressions, selects a random point, and exchanges the subexpressions behind this point.

SHARVIND is a re-targetable tool that is currently running in a Microsoft Access environment. Re-targetable means that the tool can be integrated with other database management systems, such as ORACLE, without much effort. The tool takes as input a mining question and possibly requirements (e.g., not to use certain attributes in the mining process) posed by a user. Then, a random number of expressions, called initial population, is selected. The initial population is manipulated by applying the cross-over and mutation operators. In order to compute the fitness of individuals, they are translated into corresponding SQL queries which are passed to the MS Access dbms. The fittest individuals are selected to form the next generation and the manipulation process is repeated until no significant improvement of the population can be observed. As output, the tool delivers expressions whose corresponding number of tuples falls in a user-defined interval.

We have mined two real-life databases with SHARVIND. Both databases contain aircraft incident data. The mining question that we have posed to both databases is: "What are the profiles of risky flights?" To one of these databases, we have posed some additional mining questions concerning safety aspects (see Section 7). We have presented the mining results to safety experts at our laboratory and the overall conclusion was that the answers to the mining questions were correct and promising. The mining results helped safety experts to gain insight in the databases and hopefully also knowledge in future.

### 1.1 Related work

It has been recognised by several researchers that genetic algorithms might be suitable for data mining tasks [3, 4, 8, 9, 10]. In [3, 10], a genetic approach has been proposed to learn first order logic rules and in [9], a framework is proposed for data mining based on genetic programming. In [4], a genetic algorithm is applied in the context of direct marketing. In [8], a general overview of a pattern search tool is given. Standard statistical measures are used to evaluate patterns.

The efforts in [3, 4, 8, 10] are focussed towards machine learning, and the important data mining issue of integration with databases is superficially discussed or not discussed at all. Although the framework in [9] stresses on the integration of genetic programming and databases, an elaborated approach to implement and to evaluate the framework is not presented. Furthermore, the proposed algorithms in [3, 10] are not implemented as well.

Our work distinguishes from above-mentioned efforts on the following aspects. First, we have implemented our approach and have applied it on two real-life databases. Second, we propose a re-targetable architecture, in which a genetic algorithm is integrated with databases. We note that in [4, 10] individuals are represented as binary strings or as vectors. Third, we have made a first attempt to model a fitness function such that the number of disk accesses may be optimized, which speeds up the mining session.

Other related research has been reported in [13, 14]. In these efforts, the authors use variants of a hill climber to identify the group(s) of tuples satisfying a mining question. Since a genetic-based algorithm is capable of exploring different parts of a search space, it has, by nature, a better chance to escape from a local optimum than a hill climber.

A major advantage of our approach is that we have solved the partitioning of attribute values —which is necessary for many mining algorithms— by choosing a suitable mutation operator (see Section 3). In general, data mining algorithms require a technique that partitions the domain values of an attribute in a limited set of ranges, see amongst others [17], simply because considering all possible ranges of domain values is infeasible. Finding a proper partitioning technique for attribute values is a tough problem [18].

### 1.2 A guided tour

In Section 2, we discuss the relation between searching and data mining in more detail. In Section 3, we tailor a genetic algorithm for mining purposes. Since our fitness function gives rise to optimization of the number of disk accesses, we discuss, in Section 4, a number of optimization rules. In Section 5, we discuss the overall genetic-based mining algorithm. Section 6 is devoted to the architecture and implementation of SHARVIND. In Section 7, we report on some mining results that we have obtained by mining a number of databases. Section 8 concludes the paper.

| tid | *gender* | *age* | *town* | *category* | *price* | *damage* |
|---|---|---|---|---|---|---|
| 1 | male | 20 | Rome | leased | 70K | yes |
| 2 | female | 35 | Amsterdam | not leased | 80K | yes |
| 3 | male | 24 | Amsterdam | leased | 75K | yes |
| 4 | male | 28 | Rome | not leased | 40K | yes |
| 5 | female | 28 | The Hague | leased | 50K | no |
| ... | ... | ... | ... | ... | ... | ... |

Table 1: Snapshot of the database *Driver*

## 2 Data mining and searching

In the following, a database consists of a universal relation [7]. The relation is defined over some independent single valued attributes, such as $att_1, att_2, ..., att_n$, and is a subset of the Cartesian product $\mathbf{dom}(att_1) \times \mathbf{dom}(att_2) \times ... \times \mathbf{dom}(att_n)$, in which $\mathrm{dom}(att_j)$ is the set of values that can be assumed by attribute $att_j$.

A tuple is an ordered list of attribute values to which a unique identifier, referred as tid, is attached. An expression is defined as a conjunction of predicates, and is used to select a set of tuples from the database, which in turn represents a class in the database. Consider the relation *Driver(gender, age, town, category, price, damage)* and a snapshot of this relation as depicted in Table 1. For example, the expression *gender*='male' selects the tuples corresponding to the males, i.e., the tuples 1, 3 and 4 in Table 1, the expression *age* **in** [18,24] $\wedge$ *category*='leased' selects the set of tuples corresponding to persons between eighteen and twenty four years old driving a leased car, i.e., the tuples 1 and 3, etc. A main advantage of introducing the notion of expression is that we do not have to enumerate explicitly all tuples of a class, and therefore an expression can be regarded as a summary/description of a class. So, a database contains an enormous number of classes, and each class can be represented as an expression or a disjunction of expressions.

Note, the disjunction (*gender*='male' $\wedge$ *age*=20 $\wedge$ *town*='Rome' $\wedge$ *category*='leased' $\wedge$ *price*=70K $\wedge$ *damage*='yes' ) $\vee$ ( *gender*='female' $\wedge$ *age*=35 $\wedge$ *town*='Amsterdam' $\wedge$ *category*='not leased' $\wedge$ *price*=80K $\wedge$ *damage*='yes' ) is a straightforward way to select the first and the second tuple from Table 1.

Data mining may now be regarded as the search for useful, previously unknown expressions from a space of expressions without inspecting the whole space. The search space is formed by all possible expressions with regard to a database. Note that the number of expressions grows exponentially with the number of attributes and the number of tuples in the database.

Although the usefulness of an expression generally depends on the application, we know before-hand that the following three categories of expressions will not be interesting.

1. Expressions that select zero tuples. An example of such an expression is *age*=20 $\wedge$ *age*=36. Since age is a single valued attribute, no tuples will qualify.
2. Expressions that select a set of arbitrary tuples that do not have any common characteristics. Such a set can easily be obtained by a disjunction of an arbitrary number of expressions.
3. Expressions that consist of a single predicate, called *elementary* expression. Knowledge with regard to these expressions is stored in the data dictionary, which is easily accessible.

Therefore, we discard these categories from the search space by imposing the following restrictions to expressions: 1) an attribute appears at most once in an expression, 2) disjunctions of expressions are not allowed in the search space, and 3) an expression should contain at least one conjunction.

So, the elements of the search space are expressions that satisfy above-mentioned restrictions. The challenge is to come up with search strategies that are able to find the interesting elements by inspecting a relatively small part of the search space.

## 3 Mining with genetic algorithms

In this section, we discuss the issues that play a role in tailoring a genetic algorithm for data mining. Section 3.1 is devoted to the representation of individuals in a population. Then, in Section 3.2, we discuss a fitness function, which computes the quality of an individual. Finally, in Section 3.3, we discuss the two operators that are used to manipulate an individual.

### 3.1 Representation

As stated in Section 2, an expression is a conjunction of predicates and a predicate is defined on a database attribute. An individual is defined as an expression to which some restrictions are imposed with regard to the notation of elementary expressions.

The notation of an elementary expression depends on the domain type of the involved attribute. If there exists no ordering relationship between the attribute values of an attribute *att*, we represent an elementary expression as follows: *expression* := *att* **is** $(v_1, v_2, ..., v_n)$, in which $v_i \in \mathrm{dom}(att)$, $1 \leq i \leq n$. In this way, we express that an attribute *att* assumes one of the values in the set $\{v_1, v_2, ..., v_n\}$. If an ordering relationship exists between the domain values of an attribute, an elementary expression is denoted as *expression* := *att* **in** $[v_i, v_k]$, $i \leq k$, in which $[v_i, v_k]$ represents the values within the range of $v_i$ and $v_k$.

A population is defined as a set of individuals. An example of a population, based on the relation *Driver*, is given in Figure 1.

### 3.2 Fitness function

Since a genetic algorithm is aimed to the optimization of a fitness function, this function is one of the keys to

$p_1 =$ *gender* **is** ('male') $\wedge$ *age* **in** [19,34]
$p_2 =$ *age* **in** [29,44] $\wedge$ *category* **is** ('leased') $\wedge$ *town* **is** ('Rome', 'Amsterdam', 'Cairo')
$p_3 =$ *gender* **is** ('male') $\wedge$ *age* **in** [29,34] $\wedge$ *category* **is** ('leased')
$p_4 =$ *gender* **is** ('female') $\wedge$ *age* **in** [29,40] $\wedge$ *category* **is** ('leased') $\wedge$ *price* **in** [50K, 100K]
$p_5 =$ *gender* **is** ('male') $\wedge$ *price* **in** [20K,45K]

Figure 1: Example of a population

success. Consequently, a fitness function should represent issues that play a role in the optimization of a specific problem. Before enumerating a number of issues that play a role in the context of data mining, we introduce the notions of cover.

**Definition 1:** Let $D$ be a database and $p$ an individual defined on $D$. Then, the number of tuples that satisfies the expression corresponding to $p$ is called the cover of $p$, and is denoted as $\|\sigma_p(D)\|$. The set of tuples satisfying $p$ is denoted as $\sigma_p(D)$.

Note that $p$ can be regarded as a description of a class in $D$ and $\sigma_p(D)$ summarizes the tuples satisfying $p$. Within a class we can define subclasses. In the following, we regard the classification problem as follows: *Given a target class t, search interesting subclasses, i.e., individuals, within class t.* We note that the target class is the class of tuples in which interesting knowledge should be searched for. Suppose we want to expose the profiles of risky drivers, i.e., the class of persons with (more than average) chances of causing an accident, from the database *Driver*. Then, these profiles should be searched for in a class that records the characteristics of drivers that caused accidents. Such a class may be described as *damage* = ('yes').

We feel that the following issues play a role in classification problems.

- The cover of the target class. Since results from data mining are used for informed decision making, knowledge extracted from databases should be supported by a significant part of the database. This increases the reliability of the results. So, a fitness function should take into account that small covers are undesired.

- The ratio of the cover of an individual $p$ to the cover of the target class $t$, i.e., $\frac{\|\sigma_p(D)\cap\sigma_t(D)\|}{\|\sigma_t(D)\|}$. If the ratio is close to 0, this means that only a few tuples of the target class satisfy individual $p$. This is undesired for the same reason as a small cover for a target class. If the ratio is close to 1, almost all tuples of the target class satisfy $p$. This is also undesired because this will result in knowledge that is often known. A fitness function should take these properties into account.

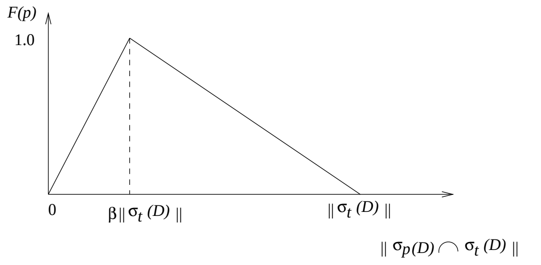

Figure 2: Shape of the fitness function

Taking into account above-mentioned issues, we have defined the following fitness function:

$$F(p) = \begin{cases} \frac{\|\sigma_p(D)\cap\sigma_t(D)\|}{\beta\|\sigma_t(D)\|}C(t) & \text{if } \|\sigma_p(D)\cap\sigma_t(D)\| \\ & \leq \beta\|\sigma_t(D)\| \\ \frac{\|\sigma_p(D)\cap\sigma_t(D)\|-\|\sigma_t(D)\|}{\|\sigma_t(D))\|(\beta-1)}C(t) & \text{otherwise} \end{cases}$$

in which $0 < \beta \leq 1$, and

$$C(t) = \begin{cases} 0 & \text{if } \frac{\|\sigma_t(D)\|}{\|\sigma(D)\|} \leq \alpha \quad \text{with } 0 \leq \alpha \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

We note that the values for $\alpha$ and $\beta$ should be defined by the user and will vary for different applications. The value of $\alpha$ defines the fraction of tuples that a target class should contain in order to be a candidate for further exploration. The value $\beta$ defines the fraction of tuples that an individual should represent within a target class in order to obtain the maximal fitness. In Figure 2, the shape of the fitness function is presented.

The fitness grows linearly with the number of tuples satisfying the description of an individual $p$ as well as satisfying a target class $t$, i.e., $\|\sigma_p(D)\cap\sigma_t(D)\|$, above a user-defined value $\alpha$, and decreases linearly with $\|\sigma_p(D)\cap\sigma_t(D)\|$ after reaching the value $\beta\|\sigma_t(D)\|$.

It should be clear that our goal is to search for those individuals that approximate a fitness of $\beta\|\sigma_t(D)\|$. Consider the target class *damage* = ('yes') that consists of 100.000 tuples. Assume that a profile is considered risky if about 30.000 out of 100.000 persons satisfy this profile. This means that $\beta \approx 0.3$. Assuming that 33.000 of the persons that caused an accident are young males, the algorithm should find individuals like *gender* **is** ('male') $\wedge$ *age* **in** [19,28].

### 3.3 Manipulation operators

In this section, we discuss the effects of the mutation and cross-over operator on an individual.

**Mutation.** As stated in the introduction of this section, a mutation modifies an individual. In defining the mutation operator, we take into account the domain type of an attribute. If there exists no ordering relationship between the domain values, then we randomly select an attribute value and replace it by another value, which can be a NULL value as well, in an expression that contains this attribute. For example, a mutation on attribute *town* of individual $p_2$ (see Figure 1) may result into $p_2' =$ *age* **in** [29,44] $\wedge$ *town* **is** ('Rome', 'The Hague', 'Cairo') $\wedge$ *category* **is** ('leased').

If there exists a relationship between the domain values of an attribute, the mutation operator acts in the case that a single value is associated with this attribute in an expression, i.e., the expression looks as *att* **is** $(v_c)$, as follows. Let $[v_b, v_e]$ be the domain of attribute *att*. In order to mutate $v_c$, we choose randomly a value $\delta_v \in [0, (v_e - v_b)\mu]$, in which $0 \leq \mu \leq 1$. The parameter $\mu$ is used to control the maximal increase or decrease of an attribute value. The mutated value $v_c'$ is defined as $v_c' = v_c + \delta_v$ or $v_c' = v_c - \delta_v$ as long as $v_c' \in [v_b, v_e]$. To handle overflow, i.e., if $v_c' \notin [v_b, v_e]$, we assume that the successor of $v_e$ is $v_b$, and, consequently the predecessor of $v_b$ is $v_e$. To compute a mutated value $v_c'$ appropriately, we distinguish between whether $v_c$ will be increased or decreased, which is randomly determined. In the case that $v_c$ is increased

$$v_c' = \begin{cases} v_c + \delta_v & \text{if } v_c + \delta_v \in [v_b, v_e] \\ v_b + \delta_v - (v_e - v_c) & \text{otherwise} \end{cases}$$

and in the case $v_c$ is decreased

$$v_c' = \begin{cases} v_c - \delta_v & \text{if } v_c - \delta_v \in [v_b, v_e] \\ v_e - \delta_v + (v_c - v_b) & \text{otherwise} \end{cases}$$

Let us consider the situation in which more than one value is associated with an attribute *att* in an expression. If a list of non successive (enumerable) values is associated with *att*, we select one of the values and compute the new value according to one of the abovementioned formulas. If a range of successive values, i.e., an interval, is associated with *att*, we select either the lower or upper bound value and mutate it. A potential disadvantage of this strategy for intervals is that an interval may be significantly enlarged, if the mutated value crosses a domain boundary. Suppose that the domain of *age* is [18,60], and we mutate the upper bound value of the expression *age* **in** $[55, 59]$, i.e., the value 59. Assuming that the value 59 is increased by 6, then 59 is mutated in the value 23. The new expression becomes *age* **in** $[23, 55]$.

We note that the partitioning of attribute values, i.e., the selection of proper intervals in an expression, is simply adjusted by the mutation operator. As noted before, partitioning of attribute values is in general a tough problem [18].

**Cross-over.** The idea behind a crossover operation is as follows; it takes as input 2 expressions, selects a random point, and exchanges the subexpressions behind this point. To illustrate this idea, we consider a relation $R(att_1, att_2, ..., att_n)$ and two expressions, $e^i$ and $e^j$, in which *all* attributes are involved. Let $e^i$ be defined as follows: $(e_1^i \wedge e_2^i \wedge e_3^i \wedge ... \wedge e_{k-1}^i \wedge e_k^i \wedge e_{k+1}^i \wedge ... \wedge e_n^i)$, in which $e_l^i$ represents an elementary expression in which attribute $att_l$ is involved. And, let $e^j$ be defined as $(e_1^j \wedge e_2^j \wedge e_3^j \wedge ... \wedge e_{k-1}^j \wedge e_k^j \wedge e_{k+1}^j \wedge ... \wedge e_n^j)$. Then, a cross-over between $e^i$ and $e^j$ at point $k$ may result into the following two expressions, namely $e^{i'} = (e_1^i \wedge e_2^i \wedge e_3^i \wedge ... \wedge e_{k-1}^i \wedge e_k^i \wedge e_{k+1}^j \wedge ... \wedge e_n^j)$ and $e^{j'} = (e_1^j \wedge e_2^j \wedge e_3^j \wedge ... \wedge e_{k-1}^j \wedge e_k^j \wedge e_{k+1}^i \wedge ... \wedge e_n^i)$.

In general, not all attributes will be involved in an expression. This may have some undesired effects for a cross-over [6]. Solutions to these effects are also discussed in [6].

## 4 Optimization rules

We discuss two propositions that may be used to prevent the exploration of unprofitable individuals. These propositions are derived from the shape of the fitness function. The complexity of a genetic-based algorithm is determined by the evaluation of the fitness function [16]. Before presenting these propositions, we introduce the notion of a similar of an individual.

**Definition 2:** Let length$(p)$ be the number of elementary expressions involved in $p$. An individual $p_{sim}$ is a *similar* of $p$ if each elementary expression of $p_{sim}$ is contained in $p$ or $p_{sim}$ contains each elementary expression of $p$ and length$(p_{sim})$ $\neq$ length$(p)$.

As stated in the foregoing, we search for individuals with high values for the fitness function $F$. For the definition of $F$, we refer to Section 3.2. Note that the computation of $F(p)$ requires the number of tuples that satisfy individual $p$. So, these tuples should be searched for and retrieved from the database, which is a costly operation [7]. Although several techniques may be used to minimize the number of retrievals from a database [5], still large amounts of tuples have to be retrieved from the database in mining applications.

In the following, two propositions will be presented that may be used to avoid the computation of fitness values of unprofitable individuals. These propositions decide if the fitness value of a similar of an individual $p$ is worse than the fitness of $p$. If this is the case, this similar can be excluded from the search process.

**Proposition 1:** Let $p_{sim}$ be a similar of $p$. If $\|\sigma_p(D) \cap \sigma_t(D)\| \leq \beta \|\sigma_t(D)\|$ and length$(p_{sim}) >$ length$(p)$ then $F(p_{sim}) \leq F(p)$.

**Proof.** From length$(p_{sim}) >$ length$(p)$ follows that $\sigma_{p_{sim}}(D) \subseteq \sigma_p(D)$. As a consequence, $\|\sigma_{p_{sim}}(D) \cap \sigma_t(D)\| \leq \|\sigma_p(D) \cap \sigma_t(D)\|$. Since $\|\sigma_p(D) \cap \sigma_t(D)\| \leq \beta \|\sigma_t(D)\|$, it follows $F(p_{sim}) \leq F(p)$. $\square$

**Proposition 2:** Let $p_{sim}$ be a similar of $p$. If $\|\sigma_p(D) \cap \sigma_t(D)\| \geq \beta \|\sigma_t(D)\|$ and length$(p_{sim}) <$ length$(p)$ then $F(p_{sim}) \leq F(p)$.

**Proof.** Similar to the proof of Proposition 1. □

Note that the propositions do not require additional retrievals from a database to decide if $F(p_{sim}) \leq F(p)$. The propositions may contribute in optimizing the search process in different ways. Due to space limitation, we discuss an application at population level.

Recall that a cross-over is applied on a mating pair and results into two offsprings. Suppose that a mutation is performed after a cross-over, and the parent and the offspring with the highest fitness values are eligible to be mutated (see Section 5). Consider an offspring $p_o$ resulted from a cross-over, and let $p_o$ be a similar of $p$, one of its parents. If we can decide that $F(p_o) \leq F(p)$, then it is efficient to mutate $p_o$. The reason is that computation on an unmutated $p_o$ will be a wasting of effort.

## 5 Algorithm

Before describing the overall genetic-based mining algorithm, we discuss a mechanism to select an individual for a next generation.

The mechanism to select individuals for a new generation is based on the technique of elitist recombination [19]. According to this technique, the individuals in a population are randomly shuffled. Then, the cross-over operation is applied on each mating pair, resulting into two offsprings. The parent and the offspring with the highest fitness value are selected for the next generation. In this way, there is a direct competition between the offsprings and their own parents.

The elitist recombination technique has been chosen for two reasons. First, there is no need to specify a particular cross-over probability, since each individual is involved in exactly one cross-over. Second, there is no need for intermediate populations in order to generate a new population as is the case in a traditional genetic algorithm. These properties simplify the implementation of a genetic algorithm. Let us outline the overall algorithm.

The algorithm starts with the initialization of a population consisting of an even number of individuals, called $P(t)$. The individuals in this population are shuffled. Then, the cross-over operation is applied on two successive individuals. After completion of a cross-over, the fitness values of the parents are compared[1]; the parent with the highest value is selected and it may be mutated with a probability $c$. This parent, $p'_{\text{sel}}$, is added to the next generation, and in case it is mutated its fitness value is computed. Then, for each offspring, $p_o$, we test if this offspring is a similar of $p'_{\text{sel}}$ and if its fitness value is worse or equal than $p'_{\text{sel}}$. If this is the case, $p_o$ is an unpromising individual, and, therefore, we always mutate $p_o$. Otherwise, we mutate $p_o$ with probability $c$. Note, to compare the fitness value between $p'_{\text{sel}}$ and $p_o$, the propositions of

[1]These values are already computed and stored by the algorithm.

```
program Genetic Algorithm;
initialize($P(t)$);
FOR $p \in P(t)$ DO $F(p)$ OD;
$F'(P(t+1)) := \epsilon + 1$; $F'(P(t)) := 0$;
WHILE $F'(P(t+1)) - F'(P(t)) \geq \epsilon$ DO
    $F'(P(t)) := F'(P(t+1)) := 0$;
    $j := 1$;
    shuffle($P(t)$);
    WHILE $j <$ *population_size* DO
        cross-over($p_j, p_{j+1}, o_1, o_2$);
        IF $F(p_j) > F(p_{j+1})$ THEN $p_{\text{sel}} := p_j$
                                  ELSE $p_{\text{sel}} := p_{j+1}$
        FI;
        mutate($p_{\text{sel}}, c, p'_{\text{sel}}$);
        IF $p'_{\text{sel}} \neq p_{\text{sel}}$ THEN $F(p'_{\text{sel}})$ FI;
        $p'_{\text{sel}} \rightarrow P(t+1)$;
        FOR $k = 1,2$ DO
            IF (similar($p'_{\text{sel}}, o_k$) AND $F(o_k) \leq F(p'_{\text{sel}})$)
                THEN mutate($o_k$, 1.0, $o'_k$);
                ELSE  mutate($o_k, c, o'_k$);
            FI;
        OD;
        $F(o'_1)$; $F(o'_2)$;
        IF $F(o'_1) > F(o'_2)$ THEN $o'_1 \rightarrow P(t+1)$
                              ELSE $o'_2 \rightarrow P(t+1)$;
        FI;
        $j := j + 2$;
    OD;
    For $p \in P(t)$ DO $F'(P(t)) := F'(P(t)) + F(p)$ OD;
    For $p \in P(t+1)$ DO
        $F'(P(t+1)) := F'(P(t+1)) + F(p)$
    OD;
    $P(t+1) := P(t)$;
OD;
END.
```

Figure 3: Sketch of the genetic-based algorithm

the previous section are used. So, no additional fitness values are computed for this comparison. After possible mutation of the offsprings, their fitness values are computed, and the fittest offspring is added to the new generation. This process is repeated for all individuals in a generation.

Once the new population has been built up, the total fitness of the existing as well as of the new population is computed, and compared. The algorithm terminates if the total fitness of the new population does not significantly improve compared with the total fitness of the existing population, i.e., that the improvement of the total fitness of the new population is less than a threshold value $\epsilon$.

In Figure 3, the algorithm is sketched. We note that the procedure cross-over($p_1, p_2, o_1, o_2$) takes two individuals $p_1$ and $p_2$ as input, applies a cross-over, and produces two offsprings $o_1$ and $o_2$. Procedure mutate($p, c, p'$) mutates an individual $p$ with probability $c$ into $p'$ and similar($p_1, p_2$) is a boolean function that decides whether two individuals are similar or not. The symbol $P(t)$ represents a population at time $t$, while $p_j$ is the $j$-th individual in $P(t)$. The total fitness of a population is defined as $F'(P(t)) = \sum_{p \in P(t)} F(p)$.

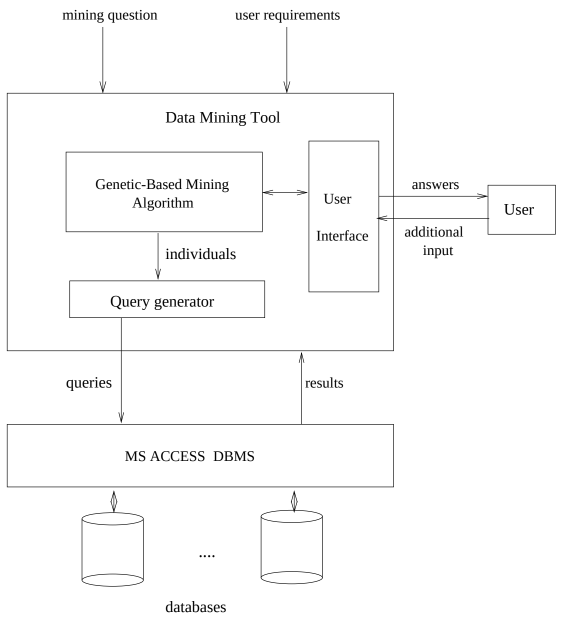

Figure 4: Architecture of SHARVIND

## 6 SHARVIND: a genetic-based data mining tool

In two successive subsections, we discuss the architecture of and some implementation issues with regard to SHARVIND.

### 6.1 Architecture

We distinguish three modules in SHARVIND, a user interface, a genetic-based mining algorithm, and a query generator. The overall architecture of SHARVIND is depicted in Figure 4. The input consists of a mining question and additional requirements that may be posed by the user. For example, a user may demand that an attribute in a database should not be involved in the mining process for reasons of privacy (e.g., income of pilots). The user is allowed to request intermediate mining results, and if desired, the user is able to modify the input. We note that this is a useful feature of a mining tool, since, in practice, a user starts a mining session with a rough idea of the information that might be interesting, and during the mining session the user more precisely specifies, based on, amongst others, the mining results obtained so far, which information should be searched for.

Once SHARVIND receives the input, it invokes the genetic-based mining algorithm. To compute the fitness value of an individual, the algorithm needs the number of tuples that satisfies the description of the individual. Therefore, the database should be interrogated. An individual should be translated into queries that are understood by the underlying dbms. This is the task of the query generator.

Since in our case an individual is an expression, the translation of an individual into an equivalent set of SQL queries is straightforward. Note that the individual already forms the WHERE clause of a SQL query. Therefore, the query generator is relatively simple. Suppose that we require the number of tuples in the database *Driver* that satisfies the description of an individual $p = gender$ **is** ('male') $\wedge$ *age* **in** [18,24] then the corresponding SQL query is: SELECT COUNT(*) FROM *Driver* WHERE (*gender* **is** 'male') AND (*age* BETWEEN 18 AND 24).

SHARVIND has as main advantages that it is extensible and re-targetable. For the time-being, SHARVIND is equipped with a genetic-based algorithm but other mining algorithms that are based on e.g., a hill climber, simulated annealing, etc., can be easily plugged in. SHARVIND is re-targetable since the tool can be coupled to other dbmss without much effort. In the case that a dbms does not understand SQL, a new module in the query generator should be built in that translates individuals in the language that is understood by the underlying dbms, while the remainder of the system can be left untouched.

### 6.2 Implementation

SHARVIND is running in a MS Access 97 environment that uses the Microsoft Jet Engine. The genetic-based algorithm is implemented in Visual Basic and consists of approximately 2000 lines of code. Our choice for this environment is mainly determined by its suitability for rapid application and the fact that the databases we want to mine are stored in MS Access.

As discussed in Section 3, major components in developing a genetic-based algorithm are the representation of individuals, manipulation operators, fitness function, and the translation of an individual into a query that is understood by the MS Access dbms.

Individuals, which are regarded as a conjunction of predicates over attributes, are implemented as binary tables. An ordered attribute, *att* is implemented as two tuples: <*att*, lower bound value> and <*att*, upper bound value>. An unordered attribute having $n$ values is implemented as $n$ tuples, having the form <*att*, value> in the table. For each individual, a binary table is built up in this way. So, the individual *gender* **is** ('male') $\wedge$ *age* **in** [18,24] is implemented as follows:

| Attribute name | Attribute value |
|---|---|
| *gender* | male |
| *age* | 18 |
| *age* | 24 |

Once the data structure of an individual was defined, the implementation of the manipulation operators was straightforward. A cross-over is obtained by selecting two tables, splitting each table into two subtables, let's say a head and a tail table. Then, the tail tables are exchanged. A mutation is obtained by deleting and/or inserting one or more tuples. Suppose that in the above-mentioned individual *gender* **is** ('male') should be mutated in *gender* **is** ('female'). This is obtained by removing the tuple *gender* **is** ('male') from the table and inserting the new tuple *gender* **is** ('female').

Due to mutations it may occur that the range interval of an attribute grows to the domain of that attribute. In such a case, the whole database will be covered by that attribute, which is undesired for the search process. To prevent this situation, we have decided that an interval corresponding to an attribute may not grow harder than a user defined threshold value, e.g., (upper bound domain value - lower bound domain value)$\times x$, in which $x \in (0, 1]$.

The implementations of the fitness function as well as the query generator were straightforward. For each generation, we keep track of the average fitness, and the individuals with the highest and lowest fitness.

## 7 Mining an artificial and two real-life databases

We give an overview of the databases that we have mined with SHARVIND and of the results that have been obtained. Section 7.1 is devoted to an artificial database, in which pre-defined knowledge was hidden. Section 7.2 is devoted to the mining of two real-life incident databases.

### 7.1 Artificial database

This database consists of the relation *Driver*. For this database 100.000 tuples have been generated of which 50% have a value ('yes') for attribute *damage*, i.e., 50% of the tuples relate to an accident. Furthermore, the following fact was hidden in this database: young men in leased cars have more than average chances of causing an accident. The goal of mining this database was to determine whether the tool is capable of finding the hidden fact. Therefore, we have set the target class as *damage* = ('yes'), and we searched for the profile of risky drivers. The expression for the hidden profile is: *age* **in** [19,24] $\wedge$ *category* **is** ('leased') $\wedge$ *gender* **is** ('male').

We have mined the database with varying initial populations, consisting of 36 individuals. The following classes of initial populations were distinguished. 1) random: this population contained a few individuals that could set the algorithm quickly on a promising route, 2) modified random: individuals that could apparently set the algorithm on a promising route were replaced by other (not promising) individuals, 3) bad converged: this population contained individuals with low fitness values.

We have observed that the algorithm usually finds near optimal solutions, i.e. profiles that look like the hidden one, in less than 1000 fitness evaluations. The differences between the hidden profile and profiles found by the algorithm (for different initial populations) were mainly caused by variations in the range of attribute *age*.

The type of the initial population plays a role in the number of fitness evaluations required to find a near optimal solution. Running the algorithm on the database with random initial populations, the algorithm was able to find a near optimal expression quite rapidly, i.e., in about 100 fitness evaluations. Starting from modified random initial populations, 300 to 400 fitness evaluations were required for a near optimal solution. Starting from bad converged initial population, 900 to 1000 fitness evaluations were required.

With regard to the settings of the parameters $\alpha$ and $\beta$, we note that appropriate values could easily be selected, since the content of the database is precisely known.

### 7.2 Two real-life databases

The first database, called ECCAIRS is located at the Joint Research Centre in Italy. Currently, this database contains serious incident and accident data that is converted from the Scandinavian accident- and incident reporting system. This database grows with approximately 4% each year. Detailed information can be found on http://eccairs-www.jrc.it, and in [11].

The second database, called the FAA database, contains aircraft incident data that have been recorded from 1978 to 1995. For example, the database contains reports of collisions between aircraft and birds while on approach to or departure from an airport. This database can be obtained from the Internet at http://www.asy.faa.gov/asp/asy_fids.asp.

Both databases are stored in MS Access and contain NULL values and redundant data. Furthermore, integrity rules (e.g., for domain values) are hardly implemented.

In two successive sections, we discuss these databases in more detail. In Section 7.2.3, we report on the mining results.

#### 7.2.1 ECCAIRS

The ECCAIRS database consists of 36 relations and about 300 attributes. Two major relations of the database are the *ACS* and the *OCCS* relations. The *ACS* relation contains information with regard to aircraft, such as manufacturer, motor, speed of the aircraft, etc., and information about the environment in which the aircraft is involved, such as weather conditions. The *OCCS* relation describes in general terms an occurrence (incident or accident) and contains general information with regard to an occurrence, for example, time and location of an occurrence, etc. The relation *ACCS* contains 5202 tuples and 186 attributes and *OCCS* contains 5202 tuples and 27 attributes. Although the number of tuples is not large, mining might be interesting due to the large number of attributes.

However, currently 17 relations do not contain any tuples, while other relations consist of tuples having many NULL values. To gain insight in the number of NULL values in each relation, we define a *filling* factor as follows:

$$filling(R) = \frac{\#values(R)}{max(values(R))} \times 100\%, \quad \text{in which}$$

$max(values(R)) = \#tuples(R) \times \#attributes(R)$. Note that $\#values(R)$ is the number of values in relation $R$, $\#tuples(R)$ is the number of tuples in $R$, etc.

It appears that $filling(ACCS) = 19\%$ which is a bad score especially for mining and $filling(OCCS) = 72\%$. In order to make the database suitable for mining we have removed:

- All attributes with less than 2000 entries filled in.
- All attributes whose values consist of natural language. Although these attributes may expose interesting knowledge, they are removed, since our algorithm is not yet able to handle them.
- Attributes that are fully functional dependent on another attribute. For example, from the latitude/longitude attribute the city name can be derived. So, city name is a redundant attribute
- Attributes with a selectivity factor[2] close to one and attributes with a very low selectivity factor, i.e., close to $\frac{1}{\#tuples(R)}$.

After performing the removals 64 attributes were left and the filling factors for $ACCS$ and $OCCS$ become 81% and 84% respectively. Then, we join these relations into a single relation.

Since the amount of tuples is relatively small, we have mined the whole joined relation.

### 7.2.2 FAA

At our laboratory, this database is implemented as a single table that is sorted on an attribute, called report number, which served as primary key. In the following, we mean by the FAA database, the database as it is implemented and filled at our laboratory.

The FAA database consists of more than 70 attributes and about 60.000 tuples. As in the case of ECCAIRS, this database contains also NULL values, redundant data, and attributes with very high and low selectivity factors. Therefore, we have cleaned this database in the same way as ECCAIRS, in order to make it suitable for mining. After cleaning 30 attributes were left and 60.000 tuples.

[2]The selectivity factor of an attribute *att* is defined as $\frac{1}{card(att)}$, in which $card(att)$ is the number of distinct values that *att* assumes.

### 7.2.3 Mining results

We have mined above-mentioned databases with different values for a number of parameters such as number of individuals of a population, average length of an individual, mutation probability, and the maximal interval to which an attribute is allowed to grow. For the influence of different parameter settings, we refer to [11] and [16]. It appears to be reasonable to set the number of individuals around 50, the average length between 2 and 5 attributes, mutation probability between 0.1 and 0.4, and the maximum interval to which an attribute may grow around 10%.

For the ECCAIRS database, the mining question was: 'What are profiles of risky flights?'. Since the amount of tuples in the ECCAIRS database is relatively small, we have mined the joined relations. So in this case, we have not defined any specific target classes. We have proposed the mining question to SHARVIND with different $\beta$ values. Recall that $\beta$ defines the fraction of tuples that an individual should represent within a target class (in this case the whole database) in order to obtain the maximal fitness. The results that we obtain were correct but most of them were trivial. For example, when we set $\beta$ to 0.2, it appears that male pilots and scattered (1/8 to 4/8) sky conditions are involved in 19% of the incidents. Although most of the results were trivial, it helped to gain insight in the database. For example, we have observed that about 50% of the database consist of incidents where no serious injury occurs and the pilot satisfies the required license. Setting $\beta$ to 0.3 delivers the following (more complex) association:
*aircraft_type* **is** ('fixed wing') $\wedge$ *power-type* **is** ('reciprocating') $\wedge$ *license_class* **is** ('required rating') $\wedge$ *highest_degree_of_injury* **is** ('none') $\rightarrow$ pilot_induced.
This rule says that pilots who hold the required licenses and are flying with fixed wing aircraft and a reciprocating power-type cause about 30% of the accidents. However, these incidents do not cause any injury.

We have mined the FAA database by posing several mining questions to SHARVIND. Our initial mining question was: search for the class of flights with (more than average) chances of causing an incident, i.e., profiles of risky flights. We have searched for this class with different input values. These searches resulted in (valid) profiles that could easily be explained by our flight safety experts. An example of such a profile is that aircraft with 1 or 2 engines are more often involved in incidents. The explanation for this profile is that these types of aircraft perform more flights.

We successively have refined our mining question into a number of questions, such as: 1) given the fact that an incident was due to operational defects not inflicted by the pilot, what is the profile of this type of incident?, 2) given the fact that an incident was due to mistakes of the pilot, what is the profile of this type of incident?, and 3) given the fact that an incident was due to improper maintenance, what is the profile of this type of incident?

The cardinality of the target classes associated with the questions posed to SHARVIND varied from 2500 to 30000 tuples and $\beta$ varied from 0.10 to 0.20.

The answers to the proposed questions were correct and most of them contained no surprise to our domain expert. In one case the tool came up with an interesting and unexpected result. In this case, we had chosen as target class = 'pilot_induced'. We had chosen to mine this target class in order to find out what type of pilots was causing incidents. The target class contains 26000 tuples. Mining the target class with $\beta = 0.15$ resulted in the following association:
*aircraft_damage* **is** ('minor') $\wedge$ *primary_flight_type* **is** ('personal') $\wedge$ *type_of_operation* **is** ('general operating rules') $\wedge$ *flight_plan* **is** ('none') $\wedge$ *pilot_rating* **is** ('no rating') $\rightarrow$ pilot_induced.
This association means that pilots without flight certificates and flight plans who are flying in private aircraft are causing more incidents than other groups. This result was on the first glance a bit strange for our safety expert, since pilots without flight certificates are not allowed to fly. After a while it appeared that the association was correct and our safety expert was able to explain the association. The pilots without certificates appeared to be students whose incidents were recorded in the FAA database as well.

## 8 Conclusions & further research

We have discussed a genetic-based algorithm that may be used for data mining. Contrary to the conventional bit string representation in genetic algorithms, we have chosen a representation that fits better in the field of databases. Furthermore, we have chosen a fitness function that is close to our intuition to rank individuals. The fitness function gives rise to an optimization of the search process.

A genetic-based algorithm for data mining has two major advantages. First, the problem of partitioning attribute values in proper ranges, which is in general a tough problem [18], could be solved by choosing a suitable mutation operator. Second, a genetic-based algorithm is able to escape a local optimum and does not pose any restrictions on the structure of a search space.

We have implemented vital parts of a prototype data mining tool that is based on a genetic algorithm. We have coupled the tool to an MS Access environment in order to mine two aircraft incident databases. Our overall conclusion is threefold. First, both databases could be significantly reduced, especially in the number of attributes, due to NULL values, text attributes, etc. Second, although both databases are significantly reduced after cleaning, we feel that data mining is a promising direction for analysing aircraft incident databases. The mining results helped safety experts to gain insight in these databases. We expect that data mining may expose knowledge from these databases in future, if more and better data will be loaded. We note that data is recently started to be loaded in the ECCAIRS database. Third, our experience is that a genetic algorithm may rapidly be implemented for data mining, yielding reasonable results. However, to build an operational tool, there is still a significant effort required.

Since many real-life databases contain many unstructured data, i.e., natural language, a topic for further research is to extend our tool with this type of data. Furthermore, to get insight in the results and performance provided by a genetic-based data mining tool, a large scale evaluation is required, which is a topic for further research as well.

**Acknowledgments** The author is grateful to W. Pelt from the Royal Netherlands Navy, who made this research possible. E. Groenheiden, L. de Penning, and M. Suurd are thanked for their implementation efforts.

## References

[1] Agrawal, R., Ghosh, S., Imielinski, T., Iyer, B., Swami, A., An Interval Classiefier for Database Mining Applications, in Proc. of the 18th VLDB Conf., 1992, 560-573.

[2] Agrawal, R., Imielinski, T., Swami, A., Database Mining: A Performance Perspective, IEEE TKDE 5(6), 1993, 914-925.

[3] Augier, S., Venturini, G., Kodratoff, Y., Learning First Order Logic Rules with a Genetic Algorithm, Int. Conf. on KDD, 1995, 21-26.

[4] Bhattacharyya, S., Direct Marketing Performance Modeling Using Genetic Algorithms, INFORMS journal on Computing, 11(3), 248-257, 1999.

[5] Choenni, R., Siebes, A., Query Optimization to Support Data Mining, in Proc. DEXA '97 8th Int. Workshop on Database and Expert Systems Applications, 1997, 658-663.

[6] Choenni, R., On the Suitability of Genetic-Based Algorithms for Data MIning. ER Workshops 1998, LNCS 1552, 55-67.

[7] Elmasri, R., Navathe, S., Fundamentals of Database Systems, The Benjamin/Cummings Publishing Comp., 1989.

[8] Flockhart, I., Radcliffe, N., A Genetic Algorithm-Based Approach to Data Mining, Int. Conf. on KDD, 1996, 299-302.

[9] Freitas, A., A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalized Rule Induction, Conf on Genetic Programming, 1997, 96-101.

[10] Giordana, A., Neri, F., Saitta, L., Botta, M., Integrating Multiple Learning Strategies in First Order Logics, ML 27(3), 1997, 209-240.

[11] Groenheiden, E., A Genetic Data Mining Feasibility Study, internal report, University of Twente, The Netherlands.

[12] Güvner, H., Akman, V., Problem Representation for Refinement, Minds and Machines 2(3), 1992, 267-282.

[13] Han, J., Cai, Y., Cerone, N., Knowledge Discovery in Databases: An Attribute-Oriented Approach, Proc. of the 18th VLDB Conf., 1992, 547-559.

[14] Holsheimer, M., Kersten, M., Architectural Support for Data Mining, AAAI-94 Worksh. on Knowl. Discovery, 217-228.

[15] Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, New York, USA.

[16] Penning, H. de, Suurd, P., NLR Genetic Search Base, internal report, University of Twente, 1998.

[17] Shafer, J., Agrawal, R., Mehta, M., SPRINT: A Scalable Parallel Clasifier for Data Mining, in Proc. 22nd Int. Conf. on VLDB, 1996, 544-555.

[18] Srikant, R., Agrawal, R., Mining Quantitative Association Rules in Large Relational Tables, in Proc. ACM SIGMOD '96 Int. Conf. on Management of Data, 1996, 1-12.

[19] Thierens, D., Goldberg, D., Elitist Recombination: an integrated selection recombination GA, in 1st IEEE Conf. on Evolutionary Computing, 1994, 508-512.